Wolverines

Eden Sherman edensam@umich.edu
Aishani Chatterjee aishanic@umich.edu aishanic18
Arka Pudota arka@umich.edu

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

The APIs/websites we planned to work with were https://www.zyxware.com/articles/4344/list-of-fortune-500-companies-and-their-websites, an API of SEO called Semrush, and yahoo finance API. With the website https://www.zyxware.com/articles/4344/list-of-fortune-500-companies-and-their-websites, we planned to obtain a list of fortune 500 companies and the corresponding company website of the number one fortune 500 company using beautiful soup. We planned to use an API of SEO called Semrush to analyze performance of the number one fortune 500 company based on the website traffic information that is collected in this API. For this we will input the desired website and the output would include the number of time visits, time span, date, etc which we will collect and use for further analysis. Most recent 100 days will be collected. Finally, for the yahoo finance API we planned to have the input be the company name and the output be the stock price frequency/date which we planned to collect and use for further analysis. We planned to collect the most recent 100 days.

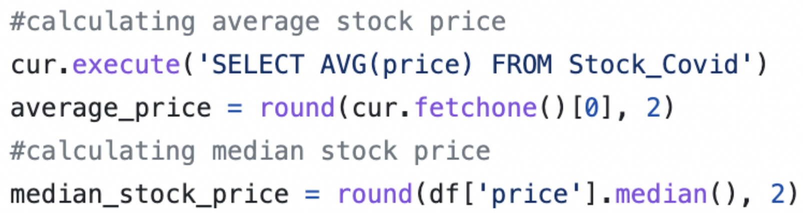
2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

We achieved the majority of these goals as we used beautifulsoup on https://www.zyxware.com/articles/4344/list-of-fortune-500-companies-and-their-websites but instead of obtaining a list of the fortune 500 companies and the corresponding company website we dug further into the html and were able to get the number one fortune 500 company as listed on the site and use that information. Since we couldn’t find a working SEO API, we instead decided to see if we could find a correlation between Covid-19 data and the stock price. Each time we ran the program, it would get 25 days of stock data, and 25 days of covid data and add each of them to an individual table in the database, named ‘Stock’ and ‘Covid’. Then we joined these 2 tables on their id and saved it in a table called ‘Stock_Covid’.

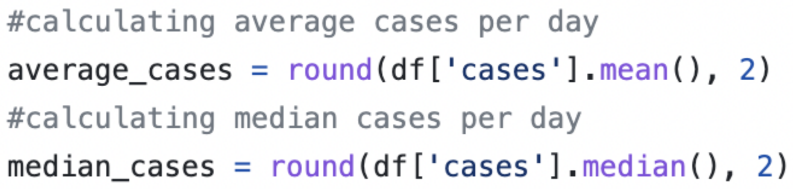
3. The problems that you faced (10 points)

The first problem we encountered was not being able to use the API of SEO called Semrush because you needed a subscription to use it, so we had to start looking for free SEO APIs. Then, we came into an issue of not being able to find any free SEO API and needed to figure out what other API data we can compare to stocks. Finally, the last problem we encountered was at first we didn't realize we needed to read the information from the API at most 25 at a time and instead we read it all at once which we needed to fix.

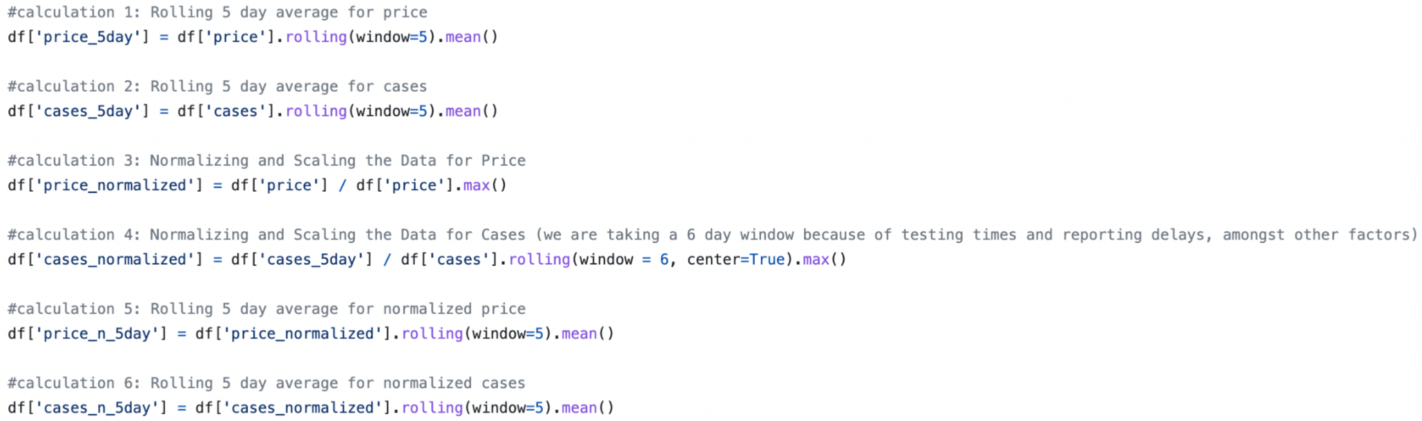
4. The calculations from the data in the database (i.e. a screenshot) (10 points)



```
#calculating average stock price
cur.execute('SELECT AVG(price) FROM Stock_Covid')
average_price = round(cur.fetchone()[0], 2)
#calculating median stock price
median_stock_price = round(df['price'].median(), 2)
```

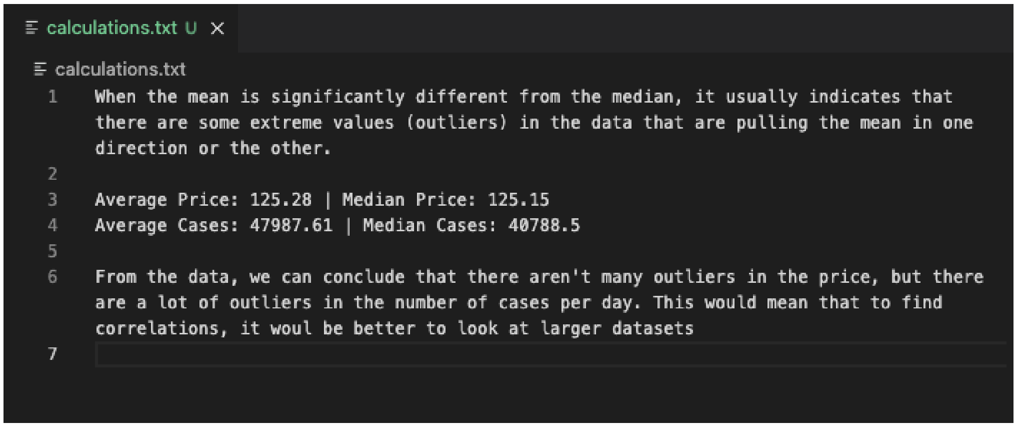


```
#calculating average cases per day
average_cases = round(df['cases'].mean(), 2)
#calculating median cases per day
median_cases = round(df['cases'].median(), 2)
```



```
#calculation 1: Rolling 5 day average for price
df['price_5day'] = df['price'].rolling(window=5).mean()

#calculation 2: Rolling 5 day average for cases
df['cases_5day'] = df['cases'].rolling(window=5).mean()

#calculation 3: Normalizing and Scaling the Data for Price
df['price_normalized'] = df['price'] / df['price'].max()

#calculation 4: Normalizing and Scaling the Data for Cases (we are taking a 6 day window because of testing times and reporting delays, amongst other factors)
df['cases_normalized'] = df['cases_5day'] / df['cases'].rolling(window = 6, center=True).max()

#calculation 5: Rolling 5 day average for normalized price
df['price_n_5day'] = df['price_normalized'].rolling(window=5).mean()

#calculation 6: Rolling 5 day average for normalized cases
df['cases_n_5day'] = df['cases_normalized'].rolling(window=5).mean()
```

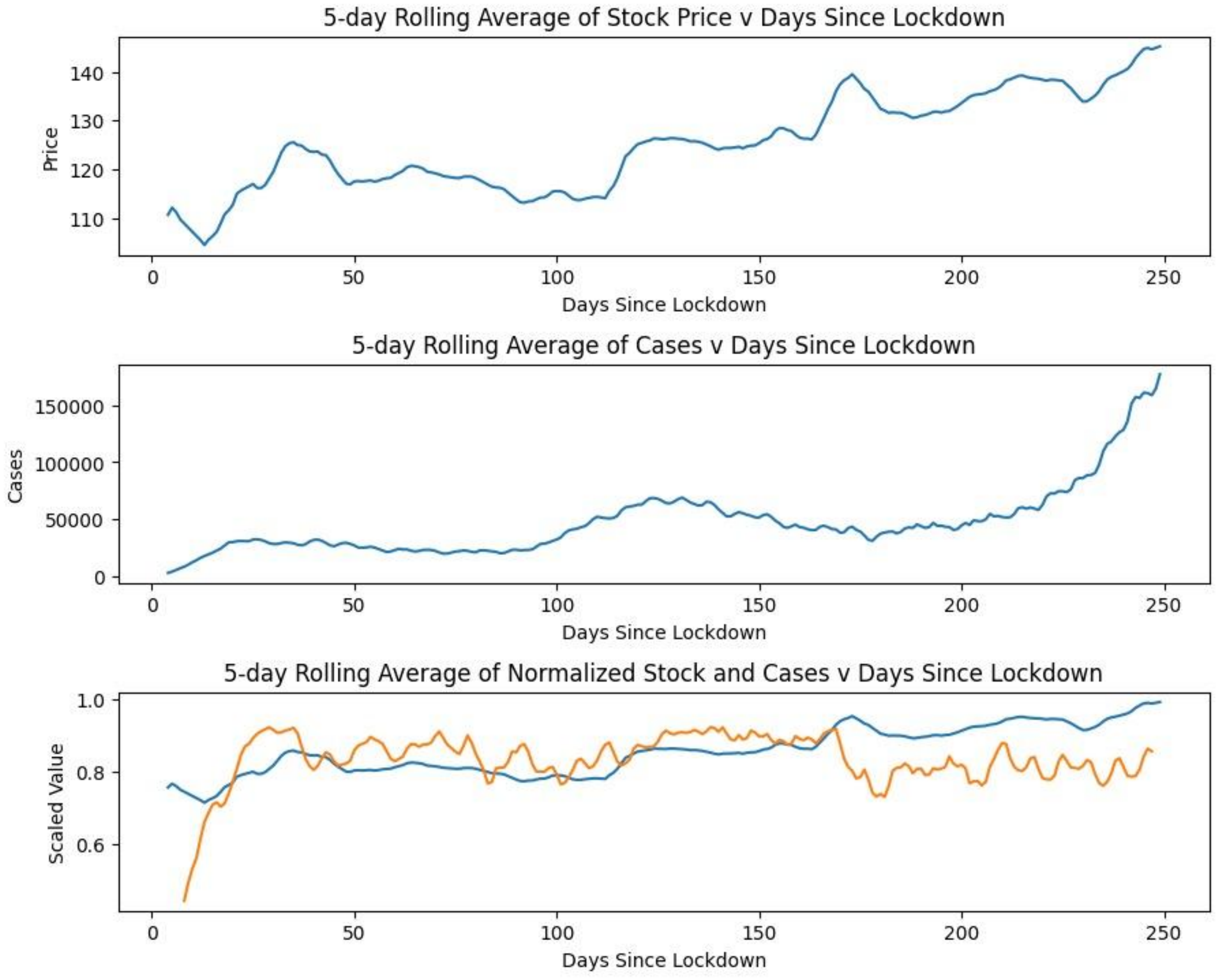
The text file output is:
(This is based on 250 days of data)



```
When the mean is significantly different from the median, it usually indicates that
there are some extreme values (outliers) in the data that are pulling the mean in one
direction or the other.

Average Price: 125.28 | Median Price: 125.15
Average Cases: 47987.61 | Median Cases: 40788.5

From the data, we can conclude that there aren't many outliers in the price, but there
are a lot of outliers in the number of cases per day. This would mean that to find
correlations, it woul be better to look at larger datasets

```

5. The visualization that you created (i.e. screen shot or image file) (10 points)

(These visualizations are on 250 days of data)


5-day Rolling Average of Stock Price v Days Since Lockdown

5-day Rolling Average of Cases v Days Since Lockdown

5-day Rolling Average of Normalized Stock and Cases v Days Since Lockdown

6. Instructions for running your code (10 points)

If this is the first time making the database, type ‘execute’ to build the tables the first time. Then, build the database by running main.py each time to add 25 rows of data from the APIs. Also, running main.py will insert the data into the databases. Next, run calc.py which will select the data from the database, do calculations on the data, write out the calculations to a file, and create the visualizations. If at any point you want to clear the data from the tables, type in execute in the input when you click run.

7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

In main.py

clean_databases(cur, conn)

Input:
cur (cursor object): A cursor object that allows Python code to execute SQL commands on a database.
conn (connection object): A connection object that represents a connection to a specific database.

Output:
None

This function drops any existing tables named 'Stock', 'Covid', and 'Stock_Covid'
from the database if they exist, and then creates new tables with the same names.

getCompanyData(url):

Input:
url: a string representing the URL of the website to scrape

Output:
A tuple containing the company name and website as strings

This function takes in a URL of a website and scrapes the webpage to find the company name and website of the first entry in the table. It then returns the company name and website as a tuple.

getTicker(company_name):

Input:
company_name: a string representing the name of the company

Output:
A string representing the stock ticker symbol of the company

This function takes in a company name and searches a CSV file containing stock ticker symbols and their corresponding companies. It returns the stock ticker symbol of the company.

getDate(cur):

Input:
cur: a sqlite3 cursor object

Output:
A string representing the latest date recorded in the Stock table of the Walmart.db database

This function takes in a cursor object and executes an SQL query to retrieve the latest date recorded in the Stock table of the Walmart.db database. It then returns this date as a string. If the Stock table is empty, it returns a default value of '2020-01-20', which represents the date of the first recorded COVID-19 case.

startAndEnd(year, month, day):

Input:
year: an integer representing the year
month: an integer representing the month
day: an integer representing the day

Output:
A tuple containing two strings representing the start and end dates, respectively, in the format 'YYYY-MM-DD'

This function takes in a year, month, and day and calculates the start and end dates for data retrieval. The start date is the input date + 1 day, and the end date is the input date + 25 days. The start and end dates are then returned as a tuple of strings.

getStockData(company_ticker, start, end):

Input:
company_ticker: a string representing the stock ticker symbol of the company
start: a string representing the start date in the format 'YYYY-MM-DD'
end: a string representing the end date in the format 'YYYY-MM-DD'

Output:
A list of tuples, with each tuple containing a string representing a date in the format 'YYYY-MM-DD' and a float representing the adjusted closing stock price for that date

This function takes in a stock ticker symbol, start date, and end date and uses the yfinance library to download the stock data for the given company between the start and end dates. It then fills in any missing data using forward filling and returns the adjusted closing prices as a list of tuples, with each tuple containing a date and price.

insertValsStock(data, cur, conn):

Input:
data: a list of tuples, with each tuple containing a string representing a date in the format 'YYYY-MM-DD' and a float representing the adjusted closing stock price for that date
cur: a sqlite3 cursor object
conn: a sqlite3 connection object

Output:
None

This function takes in a list of tuples containing stock data, a cursor object, and a connection object, and inserts the data into the Stock table of the Walmart.db database using SQL commands. It then commits the changes to the database.

getCovidData(start, end)

Input:
start (str): Start date in the format of "YYYY-MM-DD"
end (str): End date in the format of "YYYY-MM-DD"

Output:
A list of tuples containing the date (str) and the number of confirmed COVID-19 cases (int) within the given date range.

This function takes in a start date and an end date in the format of "YYYY-MM-DD" and uses the covid19dh package to retrieve the daily confirmed COVID-19 cases for the United States within the specified date range. The function then returns a list of tuples where each tuple contains the date (in the same format as the input) and the number of confirmed cases on that date.

InsertValsCovid(data, cur, conn)

Input:
data: a list of tuples, where each tuple represents a row to be inserted into the table. The first value of the tuple represents the 'date' column value, and the second value represents the 'cases' column value.
cur: a SQLite cursor object that allows execution of SQL commands.
Conn: a SQLite connection object that represents a connection to the database.

Output:
None

This function takes in a list of tuples containing stock data, a cursor object, and a connection object, and inserts the data into the Stock table of the Walmart.db database using SQL commands. It then commits the changes to the database.

main()

Input:
None

Output:
None

This function is the entry point of the program. It connects to a database, cleans the database (if requested), and then performs the following operations:

1. Retrieves the name of the top fortune 500 company and its stock ticker.

2. Retrieves a date to work from, and calculates the start and end dates for data retrieval.
3. Retrieves stock and COVID-19 data for the specified date range.
4. Inserts the retrieved data into the 'Stock' and 'Covid' tables in the database.
5. Joins the 'Stock' and 'Covid' tables to create a new table called 'Stock_Covid', which contains columns for date, stock price, and COVID-19 cases.

In calcs.py

write _to_file(f, s):

Input:
f: file object used to write a string to the file
s: string which is written to the file

Output:
None

main()

Input:
None

Output:
None

This function:

1. connects to an SQLite database named "Walmart.db".
2. Reads the data from a table called "Stock_Covid" into a Pandas dataframe.
3. Cleans the data by filling in any missing values with the previous non-null value.
4. Calculates the daily number of COVID-19 cases by taking the difference between the cumulative cases and filling in the first missing value with a fixed number.
5. Drops the cumulative cases column and renames the daily cases column for simplicity.
6. Calculates the mean and median stock prices and writes them to a file called "calculations.txt".
7. Calculates the mean and median daily cases and writes them to "calculations.txt".
8. Calculates the rolling 5-day average of stock prices and cases, as well as the normalized and scaled versions of both.
9. Plots the rolling averages and normalized data on three subplots to identify any correlations.
10. Saves the plot as "Graphs.png" and displays it.
11. Closes the file and returns a variable named "documentation", but it is not clear what this variable contains.

8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	**Issue Description**	**Location**	**Resolve (did it solve the issue?)**
04/10	Needed a subscription for SEO API	https://github.com/public-apis/public-apis	No
04/11	No SEO API worked	https://github.com/public-apis/public-apis	Yes
04/12	Beautiful soup was getting unwanted data	https://runestone.academy/ns/books/published/Win23-SI206/network/parsinghtmlusingbeautifulsoup.html	Yes
04/16	Couldn't figure out how to get date correctly	https://docs.python.org/3/library/datetime.html	Yes
04/19	Couldn’t get at most 25 at a time	https://docs.python.org/3/library/sqlite3.html	Yes